Database Systems

Department of Computer Engineering
Spring 2021

Design Report

Online Course Platform - Sapientia

Project Group 4:

- Ömer Ünlüsoy - Section 01 - 21702136
- Elif Gamze Güliter - Section 01 - 21802870
- İrem Tekin - Section 01 - 21803267
- Ece Ünal - Section 01 - 21703149

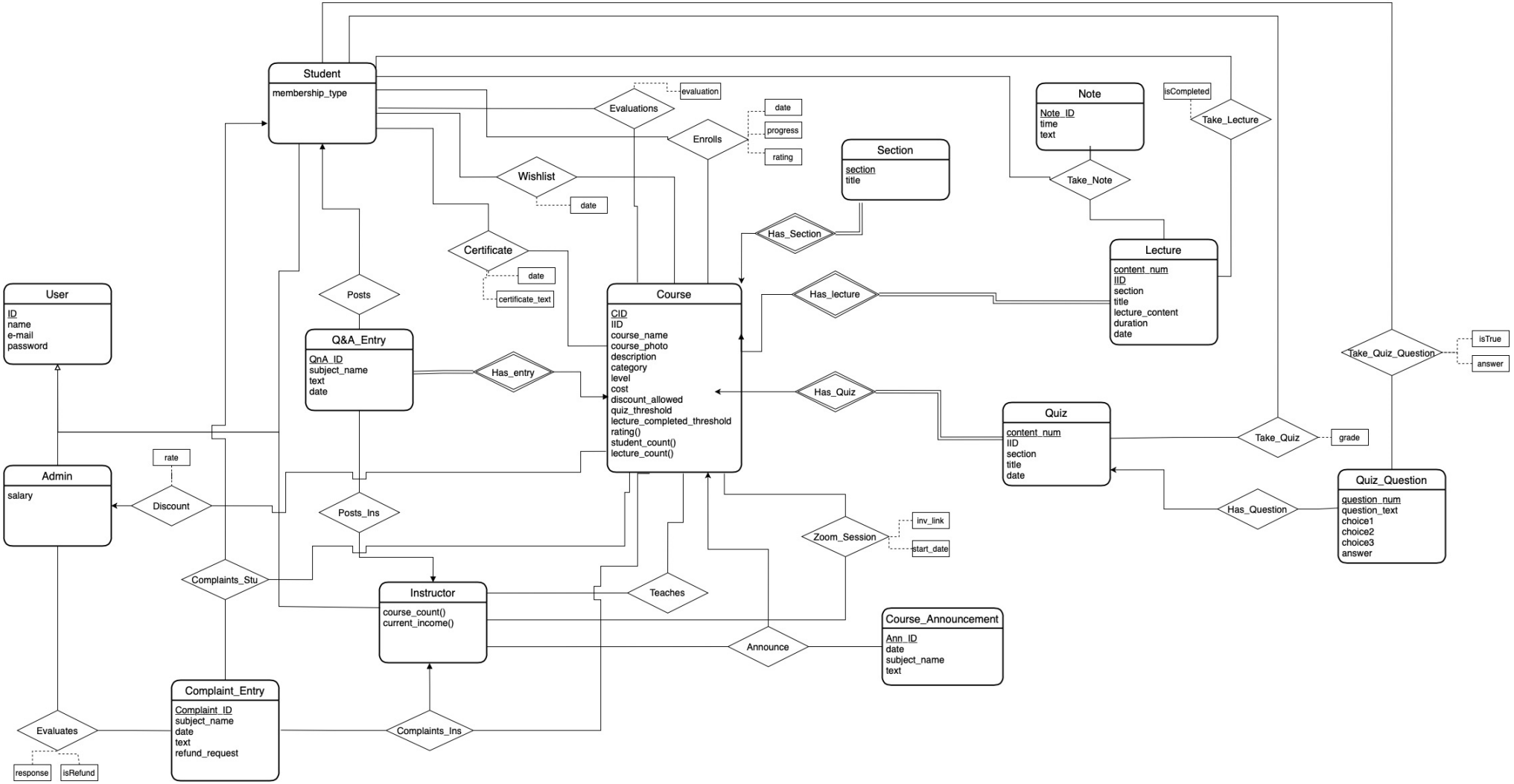
Supervisor:

- Uğur Güdükbay

Table of Contents

Revised E/R Diagram

E-R Diagram of Sapientia



Relation Schemas and SQL

Notes

General Assumptions:

ID	-> INT AUTO_INCREMENT
referencing ID	-> INT NOT NULL
short string	-> VARCHAR(30)
mid string	-> VARCHAR(50)
long string	-> VARCHAR(70)
description	-> VARCHAR(300)
money	-> NUMERIC(12, 2) OR NUMERIC(8, 2)
%	-> NUMERIC(5, 2)
date	-> DATE
URL	-> VARCHAR(1024)

Special Cases:

membership_type	-> ENUM('BRZ', 'SLV', 'GLD')
refund_request	-> BOOLEAN
duration	-> TIME
lecture_content	-> BLOB NOT NULL
membership_type	-> ENUM('BRZ', 'SLV', 'GLD') NOT NULL DEFAULT 'BRZ'
answer	-> ENUM('choice1', 'choice2', 'choice3') NOT NULL DEFAULT 'choice1'

1. Student

Relational Model:

Student(<u>SID</u>, name, e_mail, password, membership_type)

SQL Definition:

```
CREATE TABLE Student(
SID                 INT AUTO_INCREMENT,
name                VARCHAR(50) NOT NULL,
e_mail              VARCHAR(50) NOT NULL UNIQUE,
password            VARCHAR(30) NOT NULL,
membership_type     ENUM('BRZ', 'SLV', 'GLD') NOT NULL DEFAULT 'BRZ',
PRIMARY KEY (SID)
);
```

2. Instructor

Relational Model:

Instructor(IID, name, e_mail, password)

SQL Definition:

```
CREATE TABLE Instructor(
IID             INT AUTO_INCREMENT,
name            VARCHAR(50) NOT NULL,
e_mail          VARCHAR(50) NOT NULL UNIQUE,
password        VARCHAR(30) NOT NULL,
PRIMARY KEY (IID)
);
```

3. Admin

Relational Model:

Admin(AID, name, e_mail, password, salary)

SQL Definition:

```
CREATE TABLE Admin(
AID             INT AUTO_INCREMENT,
name            VARCHAR(50) NOT NULL,
e_mail          VARCHAR(50) NOT NULL UNIQUE,
password        VARCHAR(30) NOT NULL,
salary          NUMERIC(12, 2) NOT NULL DEFAULT 0.00,
PRIMARY KEY (AID)
);
```

4. Course

Relational Model:

Course(CID, IID, course_name, course_photo, description, rating, category, level, cost, discount_allowed, quiz_threshold, lecture_completed_threshold)

SQL Definition:

```
/* Assumptions:
  include functions [rating(), student_count(), lecture_count()]
  constraints for category and level
  0 <= cost <= 999,999.99
  discount_allowed = True OR False
  lecture_completed_threshold = %70
  0 <= certificate_threshold <= 100.00
  attention: FOREIGN KEY (creator_IID) REFERENCES Instructor(IID) ON DELETE NO ACTION ON
UPDATE CASCADE
```

```
        UNIQUE (creator_IID, course_name)
*/
CREATE TABLE Course(
CID                         INT AUTO_INCREMENT,
IID                         INT NOT NULL,
course_name                 VARCHAR(70) NOT NULL,
course_photo                BLOB,
description                 VARCHAR(300),
rating                      NUMERIC(5, 2) DEFAULT 0.00,
category                    VARCHAR(30),
level                       VARCHAR(30),
cost                        NUMERIC(8, 2) NOT NULL DEFAULT 0.00,
discount_allowed            BOOLEAN,
quiz_threshold              NUMERIC(5, 2),
lecture_completed_threshold         NUMERIC(5, 2),
PRIMARY KEY (CID),
FOREIGN KEY (IID) REFERENCES Instructor(IID) ON DELETE NO ACTION ON UPDATE CASCADE,
UNIQUE (IID, course_name)
);
```

5. Section

Relational Model:

Section(<u>CID</u>, <u>section</u>, title)

SQL Definition:

```
CREATE TABLE Section(
CID             INT NOT NULL,
section         INT NOT NULL,
title           VARCHAR(300),
PRIMARY KEY (CID, section),
FOREIGN KEY (CID) REFERENCES Course(CID) ON DELETE CASCADE ON UPDATE CASCADE,
INDEX (section)
);
```

6. Lecture

Relational Model:

Lecture(<u>CID</u>, <u>content_num</u>, IID, section, title, lecture_content, duration, date)

SQL Definition:

```
/* Assumptions:
  lecture_content is big data
```

lecture_content is blob (not recommended, try to store videos in file directories and hold URLs)
Attention: UNIQUE (CID, title)
*/

```
CREATE TABLE Lecture(
CID                 INT NOT NULL,
content_num         INT NOT NULL,
IID                 INT NOT NULL,
section             INT NOT NULL,
title               VARCHAR(300) NOT NULL,
lecture_content     BLOB NOT NULL,
duration            TIME NOT NULL,
date                DATE,
PRIMARY KEY (CID, content_num),
FOREIGN KEY (CID) REFERENCES Course(CID) ON DELETE CASCADE ON UPDATE CASCADE,
FOREIGN KEY (section) REFERENCES Section(section) ON DELETE CASCADE ON UPDATE
CASCADE,
FOREIGN KEY (IID) REFERENCES Instructor(IID) ON DELETE NO ACTION ON UPDATE CASCADE,
UNIQUE (CID, title),
INDEX (content_num)
);
```

7. Quiz

Relational Model:

Quiz(<u>CID</u>, <u>content_num</u>, IID, section, title, date)

SQL Definition:

```
CREATE TABLE Quiz(
CID                 INT NOT NULL,
content_num         INT NOT NULL,
IID                 INT NOT NULL,
section             INT NOT NULL,
title               VARCHAR(300) NOT NULL,
date                DATE,
PRIMARY KEY (CID, content_num),
FOREIGN KEY (CID) REFERENCES Course(CID) ON DELETE CASCADE ON UPDATE CASCADE,
FOREIGN KEY (section) REFERENCES Section(section) ON DELETE CASCADE ON UPDATE
CASCADE,
FOREIGN KEY (IID) REFERENCES Instructor(IID) ON DELETE NO ACTION ON UPDATE CASCADE,
UNIQUE (CID, title),
INDEX (content_num)
);
```

8. Quiz_Question

Relational Model:

Quiz_Question(CID, content_num, question_num, question_text, choice1, choice2, choice3, answer)

SQL Definition:

```
/*
 Attention: answer ENUM('choice1', 'choice2', 'choice3') NOT NULL DEFAULT 'choice1'
 answer = 1 => answer = choice1
 */
CREATE TABLE Quiz_Question(
CID                     INT NOT NULL,
content_num             INT NOT NULL,
question_num            INT NOT NULL,
question_text           VARCHAR(300) NOT NULL,
choice1                 VARCHAR(50) NOT NULL DEFAULT '',
choice2                 VARCHAR(50),
choice3                 VARCHAR(50),
answer                  ENUM('choice1', 'choice2', 'choice3') NOT NULL DEFAULT 'choice1',
PRIMARY KEY (CID, content_num, question_num),
FOREIGN KEY (CID) REFERENCES Course(CID) ON DELETE CASCADE ON UPDATE CASCADE,
FOREIGN KEY (content_num) REFERENCES Quiz(content_num) ON DELETE CASCADE ON
UPDATE CASCADE
);
```

9. Note

Relational Model:

Note(Note_ID, SID, content_num, time, text)

SQL Definition:

```
CREATE TABLE Note(
Note_ID                 INT AUTO_INCREMENT,
SID                     INT NOT NULL,
content_num             INT NOT NULL,
time                    TIME,
text                    VARCHAR(300) NOT NULL,
PRIMARY KEY (Note_ID),
FOREIGN KEY (SID) REFERENCES Student(SID) ON DELETE CASCADE ON UPDATE CASCADE,
FOREIGN KEY (content_num) REFERENCES Lecture(content_num) ON DELETE CASCADE ON
UPDATE CASCADE
);
```

10. Course_Announcement

Relational Model:

Course_Announcement(<u>Ann_ID</u>, IID, CID, date, subject_name, text)

SQL Definition:

```
CREATE TABLE Course_Announcement(
Ann_ID          INT AUTO_INCREMENT,
IID             INT NOT NULL,
CID             INT NOT NULL,
date            DATE,
subject_name    VARCHAR(30) NOT NULL,
text            VARCHAR(300) NOT NULL,
PRIMARY KEY (Ann_ID),
FOREIGN KEY (IID) REFERENCES Instructor(IID) ON DELETE CASCADE ON UPDATE CASCADE,
FOREIGN KEY (CID) REFERENCES Course(CID) ON DELETE CASCADE ON UPDATE CASCADE
);
```

11. QnA_Entry_Student

Relational Model:

QnA_Entry_Student(<u>QandA_ID</u>, SID, CID, subject_name, text, date)

SQL Definition:

```
CREATE TABLE QnA_Entry_Student(
QnA_ID          INT AUTO_INCREMENT,
SID             INT NOT NULL,
CID             INT NOT NULL,
subject_name    VARCHAR(30) NOT NULL,
text            VARCHAR(300) NOT NULL,
date            DATE,
PRIMARY KEY (QnA_ID),
FOREIGN KEY (SID) REFERENCES Student(SID) ON DELETE NO ACTION ON UPDATE CASCADE,
FOREIGN KEY (CID) REFERENCES Course(CID) ON DELETE CASCADE ON UPDATE CASCADE
);
```

12. QnA_Entry_Instructor

Relational Model:

QnA_Entry_Instructor(<u>QandA_ID</u>, IID, CID, subject_name, text, date)

SQL Definition:

```
CREATE TABLE QnA_Entry_Instructor(
QnA_ID      INT AUTO_INCREMENT,
IID         INT NOT NULL,
CID         INT NOT NULL,
```

```
subject_name      VARCHAR(30) NOT NULL,
text              VARCHAR(300) NOT NULL,
date              DATE,
PRIMARY KEY(QnA_ID),
FOREIGN KEY (IID) REFERENCES Instructor(IID) ON DELETE NO ACTION ON UPDATE CASCADE,
FOREIGN KEY (CID) REFERENCES Course(CID) ON DELETE CASCADE ON UPDATE CASCADE
);
```

13. Complaint_Entry_Student

Relational Model:

Complaint_Entry_Student(<u>Complaint_ID</u>, SID, CID, subject_name, text, date, refund_request)

SQL Definition:

```
CREATE TABLE Complaint_Entry_Student(
Complaint_ID        INT AUTO_INCREMENT,
SID                 INT NOT NULL,
CID                 INT NOT NULL,
subject_title       VARCHAR(30) NOT NULL,
text                VARCHAR(300) NOT NULL,
date                DATE,
refund_request      BOOLEAN,
PRIMARY KEY (Complaint_ID),
FOREIGN KEY (SID) REFERENCES Student(SID) ON DELETE CASCADE ON UPDATE CASCADE,
FOREIGN KEY (CID) REFERENCES Course(CID) ON DELETE CASCADE ON UPDATE CASCADE
);
```

14. Complaint_Entry_Instructor

Relational Model:

Complaint_Entry_Instructor(<u>Complaint_ID</u>, SID, CID, subject_name, text, date)

SQL Definition:

```
CREATE TABLE Complaint_Entry_Instructor(
Complaint_ID        INT AUTO_INCREMENT,
IID                 INT NOT NULL,
CID                 INT NOT NULL,
subject_name        VARCHAR(30) NOT NULL,
text                VARCHAR(300) NOT NULL,
date                DATE,
PRIMARY KEY (Complaint_ID),
FOREIGN KEY (IID) REFERENCES Instructor(IID) ON DELETE CASCADE ON UPDATE CASCADE,
FOREIGN KEY (CID) REFERENCES Course(CID) ON DELETE CASCADE ON UPDATE CASCADE
);
```

15. Wishlist

Relational Model:

Wishlist(SID, CID, date)

SQL Definition:

```
/*
 Binary relationship between Student and Course
 Attributes: date
 */
CREATE TABLE Wishlist(
SID             INT NOT NULL,
CID             INT NOT NULL,
date            DATE,
PRIMARY KEY (SID, CID),
FOREIGN KEY (SID) REFERENCES Student(SID) ON DELETE CASCADE ON UPDATE CASCADE,
FOREIGN KEY (CID) REFERENCES Course(CID) ON DELETE CASCADE ON UPDATE CASCADE
);
```

16. Enrolls

Relational Model:

Enrolls(SID, CID, progress, rating, date)

SQL Definition:

```
/*
 Binary relationship between Student and Course
 Attributes: progress, rating
 */
CREATE TABLE Enrolls(
SID             INT NOT NULL,
CID             INT NOT NULL,
progress        NUMERIC(5, 2) DEFAULT 0.00,
rating          NUMERIC(5, 2),
date            DATE,
PRIMARY KEY (SID, CID),
FOREIGN KEY (SID) REFERENCES Student(SID) ON DELETE CASCADE ON UPDATE CASCADE,
FOREIGN KEY (CID) REFERENCES Course(CID) ON DELETE CASCADE ON UPDATE CASCADE
);
```

17. Certificate

Relational Model:

Certificate(SID, CID, certificate_text, date)

SQL Definition:

```
/*
 Binary relationship between Student and Course
 Attributes: date
 */
CREATE TABLE Certificate(
SID                     INT NOT NULL,
CID                     INT NOT NULL,
certificate_text        VARCHAR(300),
date                    DATE,
PRIMARY KEY (SID, CID),
FOREIGN KEY (SID) REFERENCES Student(SID) ON DELETE CASCADE ON UPDATE CASCADE,
FOREIGN KEY (CID) REFERENCES Course(CID) ON DELETE CASCADE ON UPDATE CASCADE
);
```

18. Teaches

Relational Model:

Teaches(IID, CID)

SQL Definition:

```
/*
 Binary relationship between Instructor and Course
 */
CREATE TABLE Teaches(
IID             INT NOT NULL,
CID             INT NOT NULL,
PRIMARY KEY (IID, CID),
FOREIGN KEY (IID) REFERENCES Instructor(IID) ON DELETE CASCADE ON UPDATE CASCADE,
FOREIGN KEY (CID) REFERENCES Course(CID) ON DELETE CASCADE ON UPDATE CASCADE
);
```

19. Evaluates_Complaint_Entry_Student

Relational Model:

Evaluates_Complaint_Entry_Student(AID, Complaint_ID, response, isRefund)

SQL Definition:

```
/*
 Binary relationship between Admin and Complaint_Entry
```

```
 Attention: FOREIGN KEY (AID) REFERENCES Admin(AID) ON DELETE NO ACTION ON UPDATE
CASCADE
 */
CREATE TABLE Evaluates_Complaint_Entry_Student(
AID                     INT NOT NULL,
Complaint_ID            INT NOT NULL,
response                VARCHAR(300),
isRefund                BOOLEAN,
PRIMARY KEY (AID, Complaint_ID),
FOREIGN KEY (AID) REFERENCES Admin(AID) ON DELETE NO ACTION ON UPDATE CASCADE,
FOREIGN KEY (Complaint_ID) REFERENCES Complaint_Entry_Student(Complaint_ID) ON DELETE
CASCADE ON UPDATE CASCADE
);
```

20. Evaluates_Complaint_Entry_Instructor

Relational Model:

Evaluates_Complaint_Entry_Instructor(AID, Complaint_ID, response)

SQL Definition:

```
/*
 Binary relationship between Admin and Complaint_Entry
 Attention: FOREIGN KEY (AID) REFERENCES Admin(AID) ON DELETE NO ACTION ON UPDATE
CASCADE
 */
CREATE TABLE Evaluates_Complaint_Entry_Instructor(
AID                     INT NOT NULL,
Complaint_ID            INT NOT NULL,
response                VARCHAR(300),
PRIMARY KEY (AID, Complaint_ID),
FOREIGN KEY (AID) REFERENCES Admin(AID) ON DELETE NO ACTION ON UPDATE CASCADE,
FOREIGN KEY (Complaint_ID) REFERENCES Complaint_Entry_Instructor(Complaint_ID) ON
DELETE CASCADE ON UPDATE CASCADE
);
```

21. Discount

Relational Model:

Discount(AID, CID, rate)

SQL Definition:

```
/*
 Binary relationship between Admin and Course
 Attributes: rate (e.g. %30)
```

```
 Attention: FOREIGN KEY (AID) REFERENCES Admin(AID) ON DELETE NO ACTION ON UPDATE
CASCADE
 */
CREATE TABLE Discount(
AID             INT NOT NULL,
CID             INT NOT NULL,
rate            NUMERIC(5, 2),
PRIMARY KEY (AID, CID),
FOREIGN KEY (AID) REFERENCES Admin(AID) ON DELETE NO ACTION ON UPDATE CASCADE,
FOREIGN KEY (CID) REFERENCES Course(CID) ON DELETE CASCADE ON UPDATE CASCADE
);
```

22. Zoom_Session

Relational Model:

Zoom_Session(<u>Session_ID</u>, IID, CID, invite_link, start_date);

```
/*
 Binary relationship between Instructor and Course
 Attributes: invite_link, start_date
 Attention: it uses its own key to allow instructors to create several links
 */
CREATE TABLE Zoom_Session(
Session_ID              INT AUTO_INCREMENT,
IID                     INT NOT NULL,
CID                     INT NOT NULL,
invite_link             VARCHAR(1024),
start_date              TIMESTAMP,
PRIMARY KEY (Session_ID),
FOREIGN KEY (IID) REFERENCES Instructor(IID) ON DELETE CASCADE ON UPDATE CASCADE,
FOREIGN KEY (CID) REFERENCES Course(CID) ON DELETE CASCADE ON UPDATE CASCADE
);
```

SQL Definition:

23. Take_Quiz

Relational Model:

Take_Quiz(<u>SID</u>, <u>CID</u>, <u>content_num</u>, grade)

SQL Definition:

```
/*
 Binary relationship between Student and Quiz
 Attributes: grade (updated according to Student_Take_Quiz_Question)
 Attention: its key is Student(SID) U Quiz(CID, QID)
```

```
 */
CREATE TABLE Take_Quiz(
SID                     INT NOT NULL,
CID                     INT NOT NULL,
content_num             INT NOT NULL,
grade                   NUMERIC(5, 2),
PRIMARY KEY (SID, CID, content_num),
FOREIGN KEY (SID) REFERENCES Student(SID) ON DELETE CASCADE ON UPDATE CASCADE,
FOREIGN KEY (CID) REFERENCES Course(CID) ON DELETE CASCADE ON UPDATE CASCADE,
FOREIGN KEY (content_num) REFERENCES Quiz(content_num) ON DELETE CASCADE ON
UPDATE CASCADE
);
```

24. Take_Quiz_Question

Relational Model:

Take_Quiz_Question(<u>SID</u>, <u>CID</u>, <u>content_num</u>, <u>question_num</u>, answer, isTrue)

SQL Definition:

```
/*
 Binary relationship between Student and Quiz_Question
 keeps students' answers to each quiz question
 Attributes: isTrue
 */
CREATE TABLE Take_Quiz_Question(
SID                     INT NOT NULL,
CID                     INT NOT NULL,
content_num             INT NOT NULL,
question_num            INT NOT NULL,
answer                  ENUM('choice1', 'choice2', 'choice3') NOT NULL,
isTrue                  BOOLEAN,
PRIMARY KEY (SID, CID, content_num, question_num),
FOREIGN KEY (SID) REFERENCES Student(SID) ON DELETE CASCADE ON UPDATE CASCADE,
FOREIGN KEY (CID, content_num, question_num) REFERENCES Quiz_Question(CID,
content_num, question_num) ON DELETE CASCADE ON UPDATE CASCADE
);
```

25. Take_Lecture

Relational Model:

Take_Quiz_Lecture(SID, CID, content_num, isCompleted)

SQL Definition:

```
/*
 Binary relationship between Student and Lecture
 keeps students' completion to each lecture (lecture_completed_threshold in lecture (e.g. %70 for each lecture))
 Attributes: isCompleted
 */
CREATE TABLE Take_Lecture(
SID                     INT NOT NULL,
CID                     INT NOT NULL,
content_num             INT NOT NULL,
isCompleted             BOOLEAN,
PRIMARY KEY (SID, CID, content_num),
FOREIGN KEY (SID) REFERENCES Student(SID) ON DELETE CASCADE ON UPDATE CASCADE,
FOREIGN KEY (CID, content_num) REFERENCES Lecture(CID, content_num) ON DELETE CASCADE ON UPDATE CASCADE
);
```

26. Evaluations

Relational Model:

Evaluations(SID, CID, evaluation)

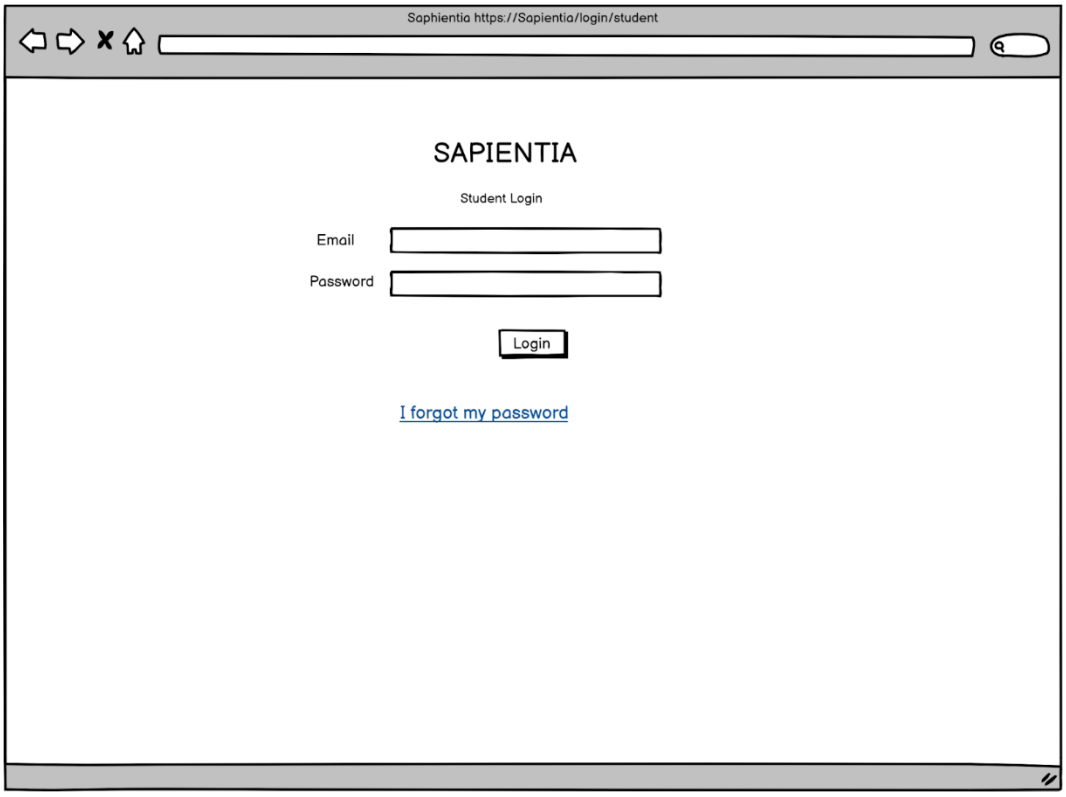
SQL Definition:

```
CREATE TABLE Evaluations(
SID             INT NOT NULL,
CID             INT NOT NULL,
evaluation      VARCHAR(300),
PRIMARY KEY (SID, CID),
FOREIGN KEY (SID) REFERENCES Student(SID) ON DELETE CASCADE ON UPDATE CASCADE,
FOREIGN KEY (CID) REFERENCES Course(CID) ON DELETE CASCADE ON UPDATE CASCADE
);
```

User Interface Design and Corresponding SQL Statements

1. Login

Student Login



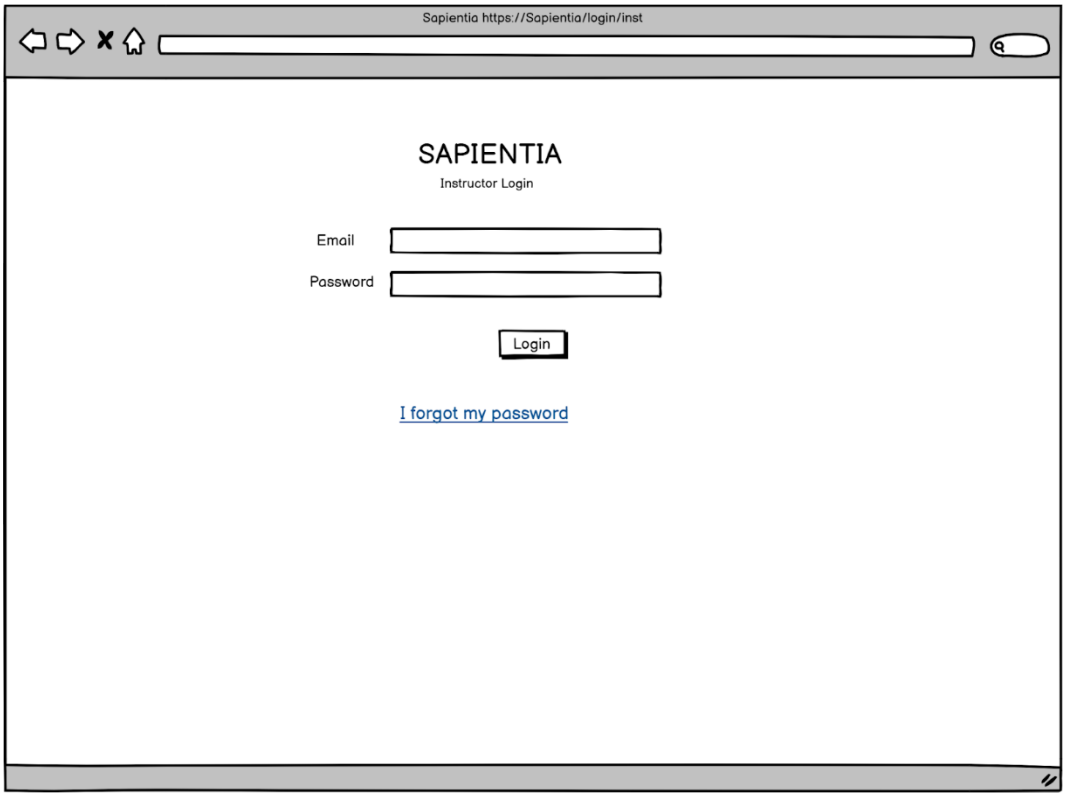
Students will log-in to Sapientia using their email and password.

Inputs: @e_mail , @password

On Login button pressed:

```
SELECT *
FROM Student
WHERE e_mail = @e_mail AND password = @password
```

Instructor Login



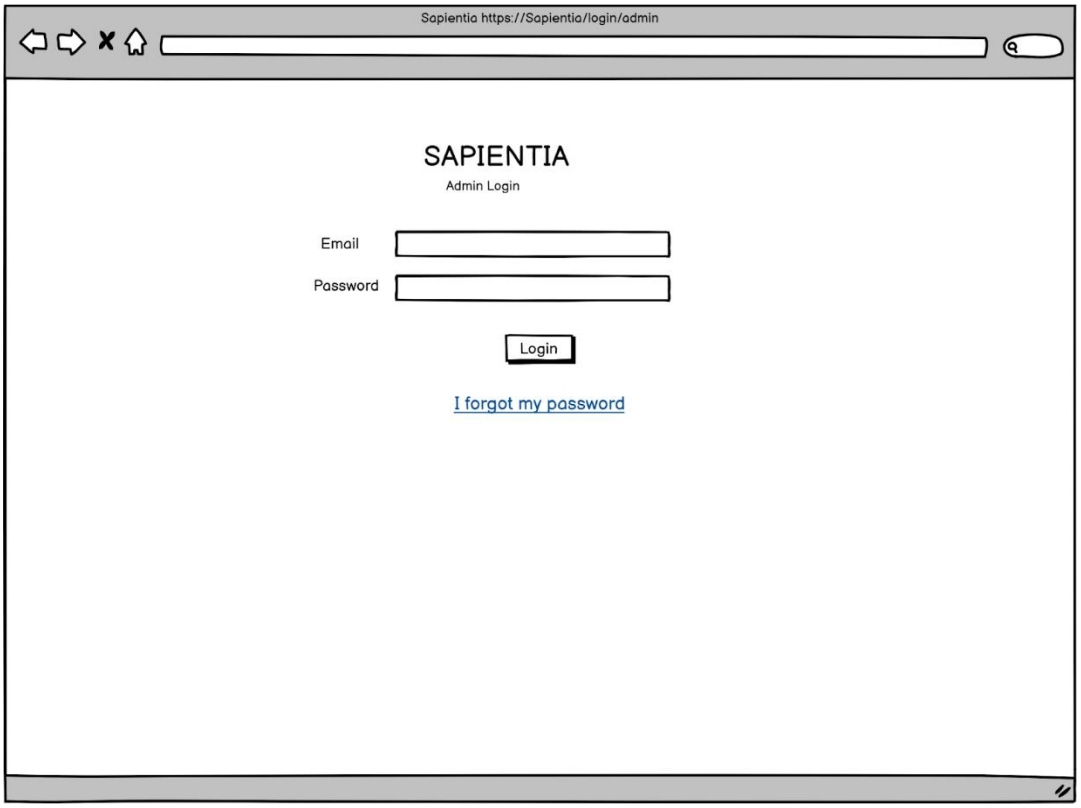
Instructors will log-in to Sapientia using their email and password.

Inputs: @e_mail , @password

On Login button pressed:

```
SELECT *
FROM Instructor
WHERE e_mail = @e_mail AND password = @password
```

Admin login



Admin will log-in to Sapientia using the email and password associated with the admin role.

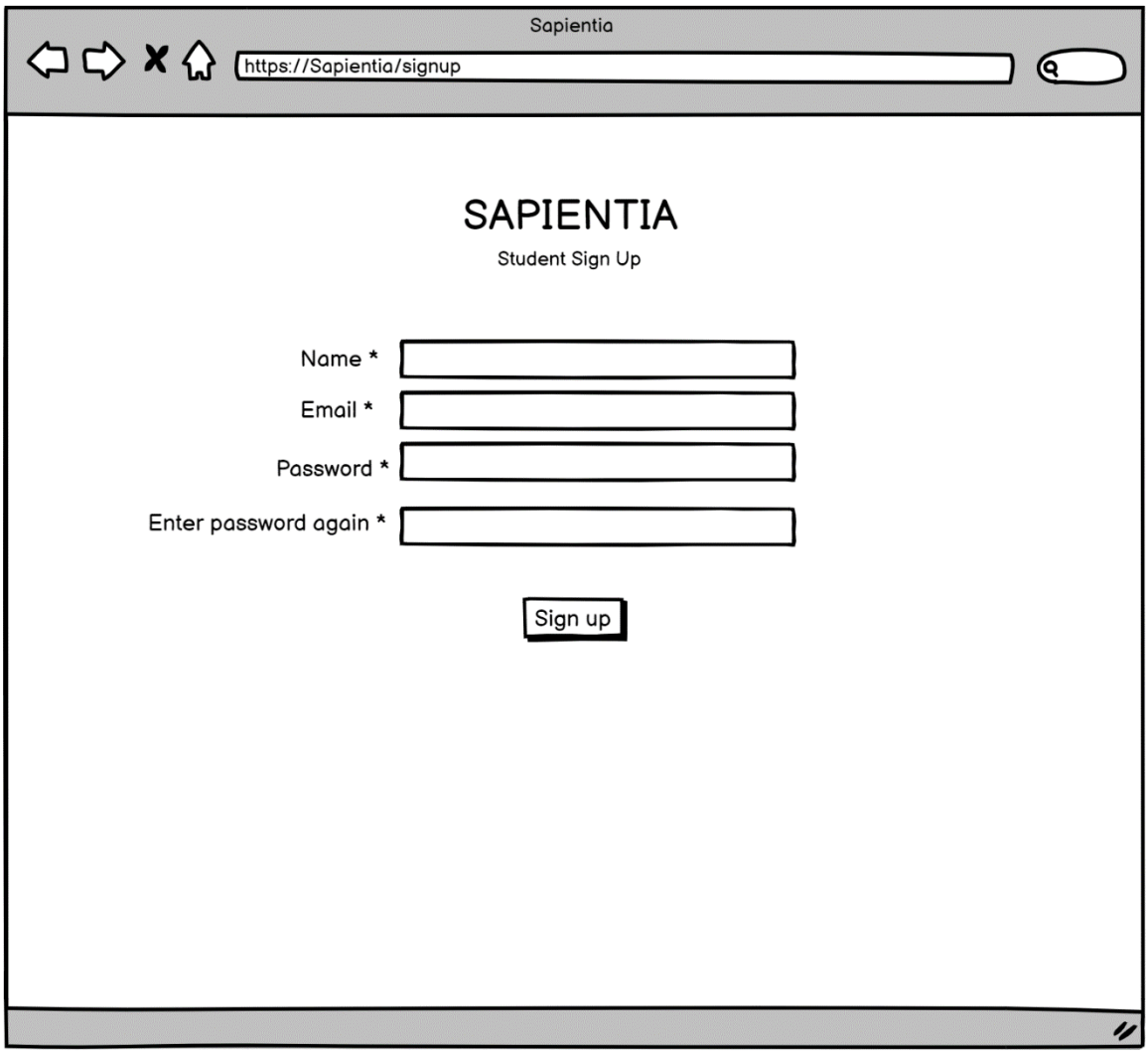
Inputs: @e_mail , @password

On Login button pressed:

SELECT *
FROM Admin
WHERE e_mail = @e_mail AND password = @password

2. Sign Up

Student Sign Up



Students will sign up by their name, email and password. The password will be checked twice.

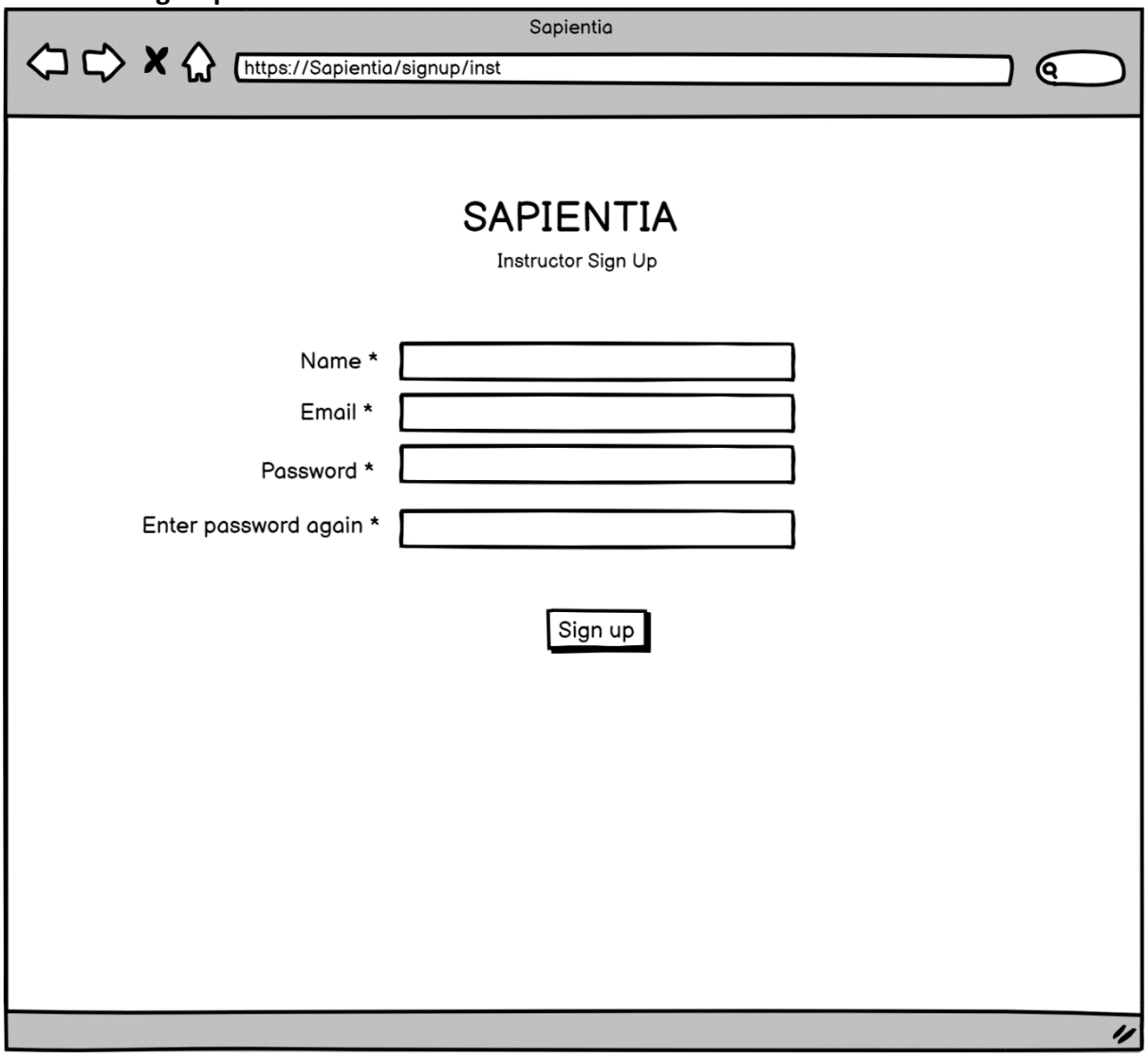
Inputs: @name, @e_mail, @password

On Sign up button pressed:

```
BEGIN
        IF NOT EXISTS ( SELECT * FROM Student
                                WHERE e_mail = @e_mail)
```

```
        BEGIN
                INSERT INTO Student (name, e_mail, password)
                VALUES (@name, @e_mail, @password)
        END
END
```

Instructor Sign Up



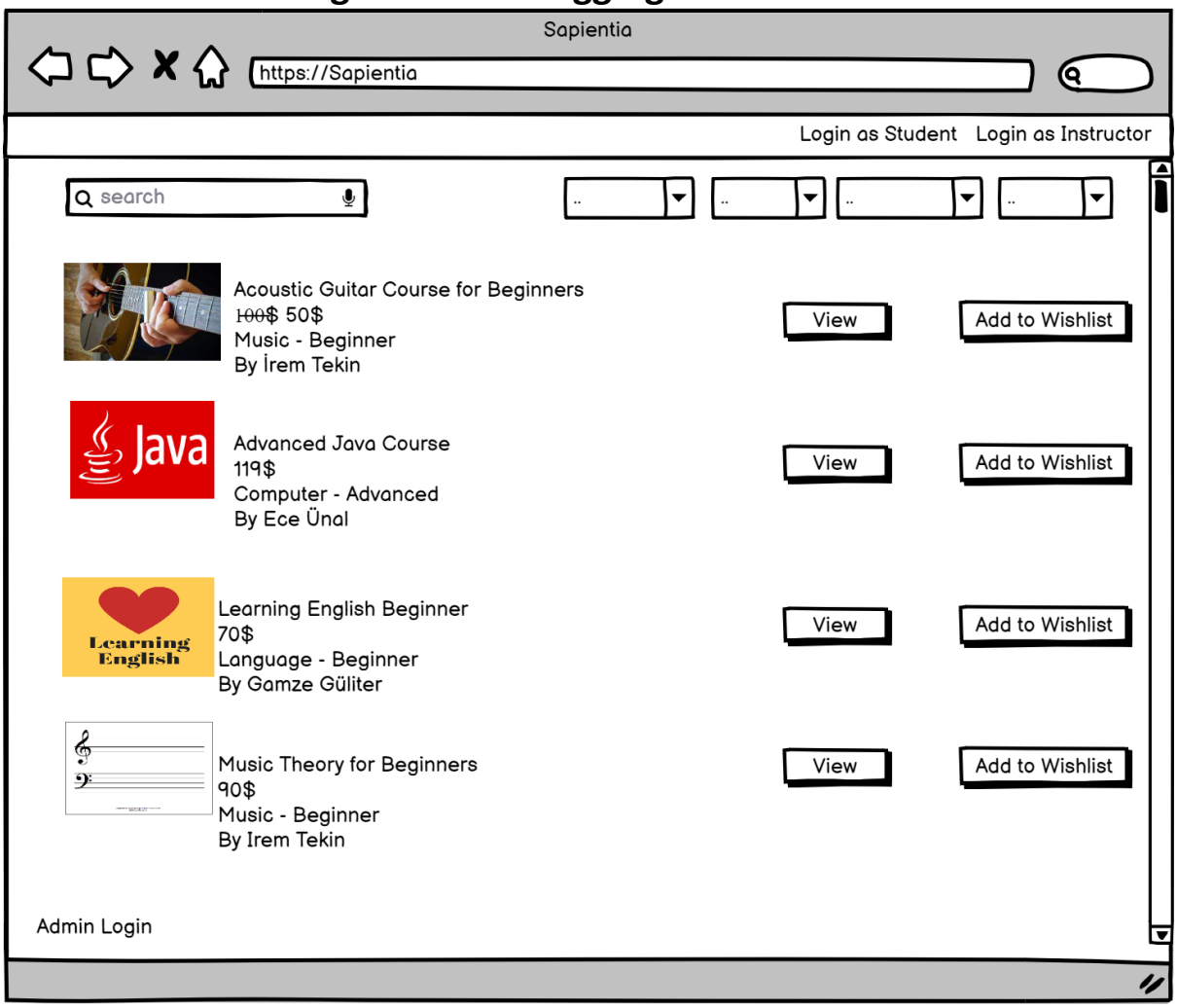
Instructors will sign up by their name, email and password. The password will be checked twice.

Inputs: @name, @e_mail, @password

On Sign up button pressed:

```
BEGIN
        IF NOT EXISTS ( SELECT * FROM Student
                                WHERE e_mail = @e_mail)
        BEGIN
                INSERT INTO Instructor (name, e_mail, password)
                VALUES (@name, @e_mail, @password)
        END
END
```

3. Main Page Without Logging in

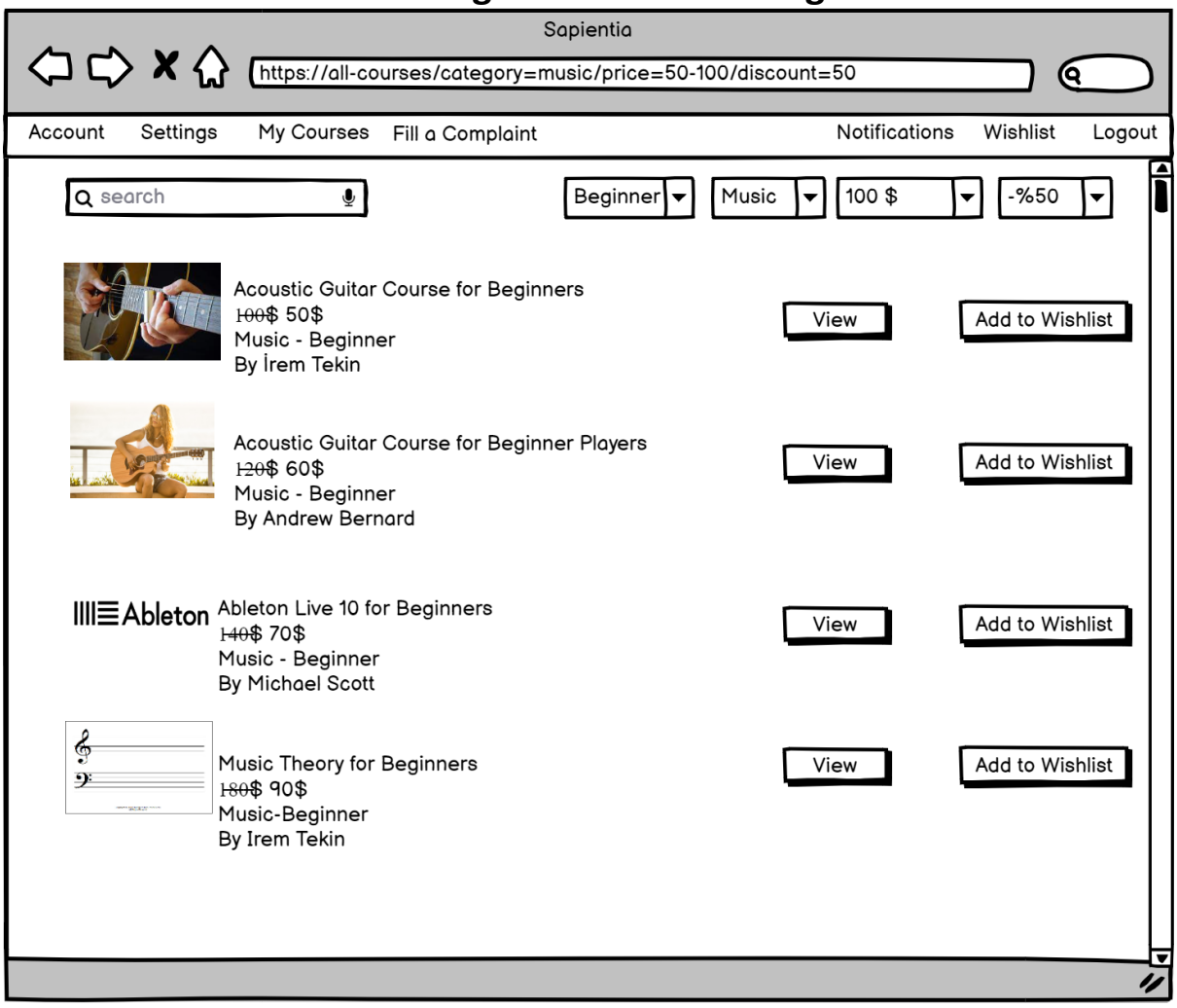


Students will be able to see courses in Sapientia without logging in. However, for a student to add a course to the wishlist or purchase it , logging in is needed.

Inputs: @category, @price_threshold, @discount, @keyword, @level

```
SELECT course_name, course_photo, cost, category, level, name
FROM Course FULL OUTER JOIN Discount NATURAL JOIN Instructor
WHERE category = @category AND price < @price_threshold
        AND rate = @discount
        AND ( course_name like '%keyword%'
            OR instructor_name like '%keyword%' )
```

4. Filtered Main Page After Student Login



When students log in to their account, they will be able to find courses to enroll. At first , the most popular courses based on the enrolled students will be shown in their main page. However, students will be able to filter courses in their main page by a keyword or level of course, the category , maximum price and the discount rate.

Case: In the above example, the user wanted to see the courses in the beginner level of music category. In addition the courses are under 100$ and have the discount rate of %50.

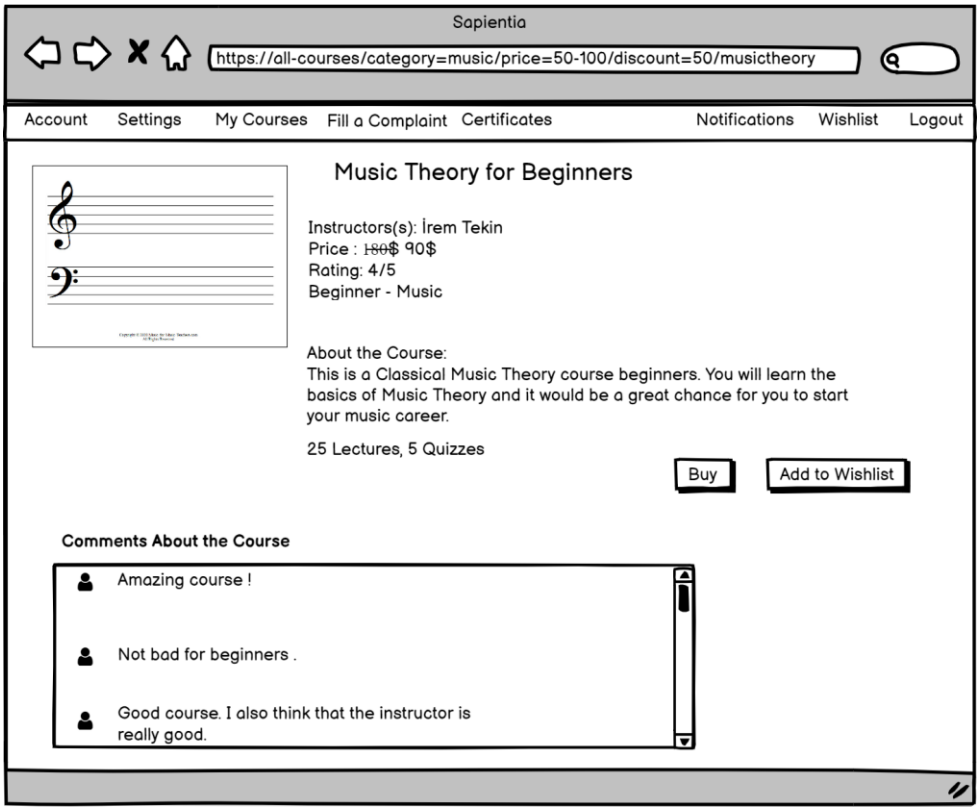
Inputs: @category, @price_threshold, @discount, @keyword, @level

```
SELECT course_name, course_photo, cost, category, level, name
FROM Course FULL OUTER JOIN Discount NATURAL JOIN Instructor
WHERE category = @category AND price < @price_threshold
        AND rate = @discount
        AND ( course_name like '%keyword%'
                OR instructor_name like '%keyword%' )
       EXCEPT ( SELECT course_name, course_photo, cost, category, level
        FROM Enrolls NATURAL JOIN Course
        WHERE SID = @student_id  )
```

On Add to Wishlist button pressed:

```
DECLARE @current_date date = getdate()
INSERT INTO Wishlist(SID, CID, date)
VALUES(@SID, @CID, @current_date)
```

5. Course Information Page



The users will be able to view information about the course if they press the **View** button shown in the right side of the course name in the main page. In this page, they will see the description of the course, price, overall rating, information such as quiz-lecture number of course's difficulty level and the comments from other students who finished the course.In addition, they can buy the course from this page and add it to their wishlist.

Case: In the above example, the user is viewing Music Theory for Beginner's course's page.

```
SELECT name, IID
FROM Teaches JOIN USING (IID) Instructor
WHERE CID = @CID

SELECT course_photo,course_name, old_cost, cost, description, avg_rating, name
FROM Course, Temp
WHERE CID = @CID

SELECT COUNT(*)
FROM Lecture
WHERE CID = @CID

SELECT COUNT(*)
FROM Quiz
WHERE CID = @CID
```

On Buy button pressed:

```
INSERT INTO ENROLLS(SID, CID, date)
VALUES (@SID, @CID, @date)


SELECT COUNT(*)
FROM Enrolls
WHERE SID = @SID
```

Using this count, we will determine membership type with PHP and then update student table.

```
UPDATE Student
SET membership_type = @type
```

On Add to Wishlist button pressed:

```
DECLARE @current_date date = getdate()
INSERT INTO Wishlist(SID, CID, date)
VALUES(@SID, @CID, @current_date)
```

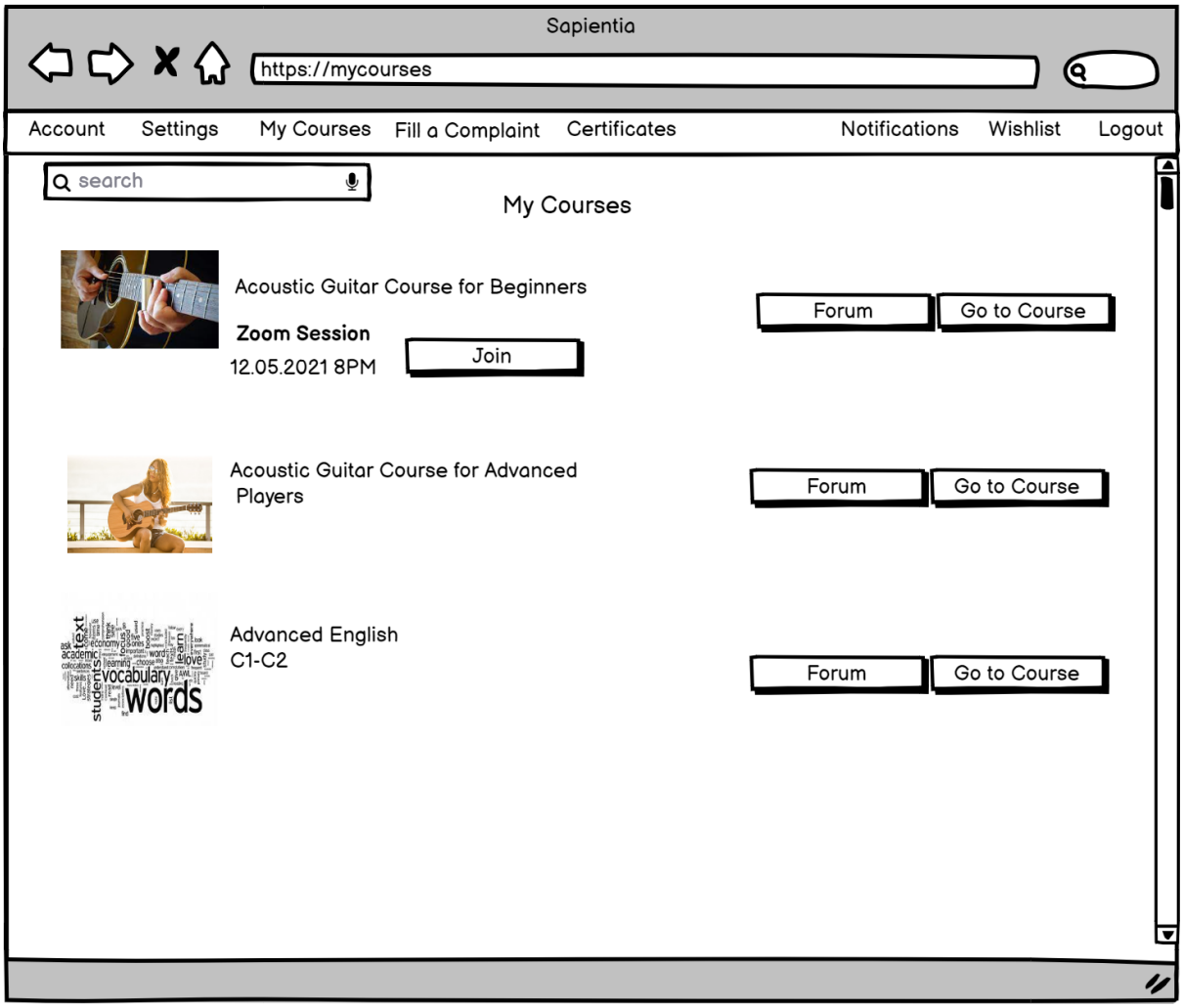
6. Wishlist

The students will be able to see their wishlist from the **Wishlist** button in the navigation bar, then they can either buy the course or remove it from the wishlist.

Case: In the above example the student added three courses to his/her wishlist.

SELECT course_name, course_photo, cost, category, level, name
FROM Course FULL OUTER JOIN Discount NATURAL JOIN Instructor NATURAL JOIN Wishlist
WHERE SID = @SID

7. My Courses

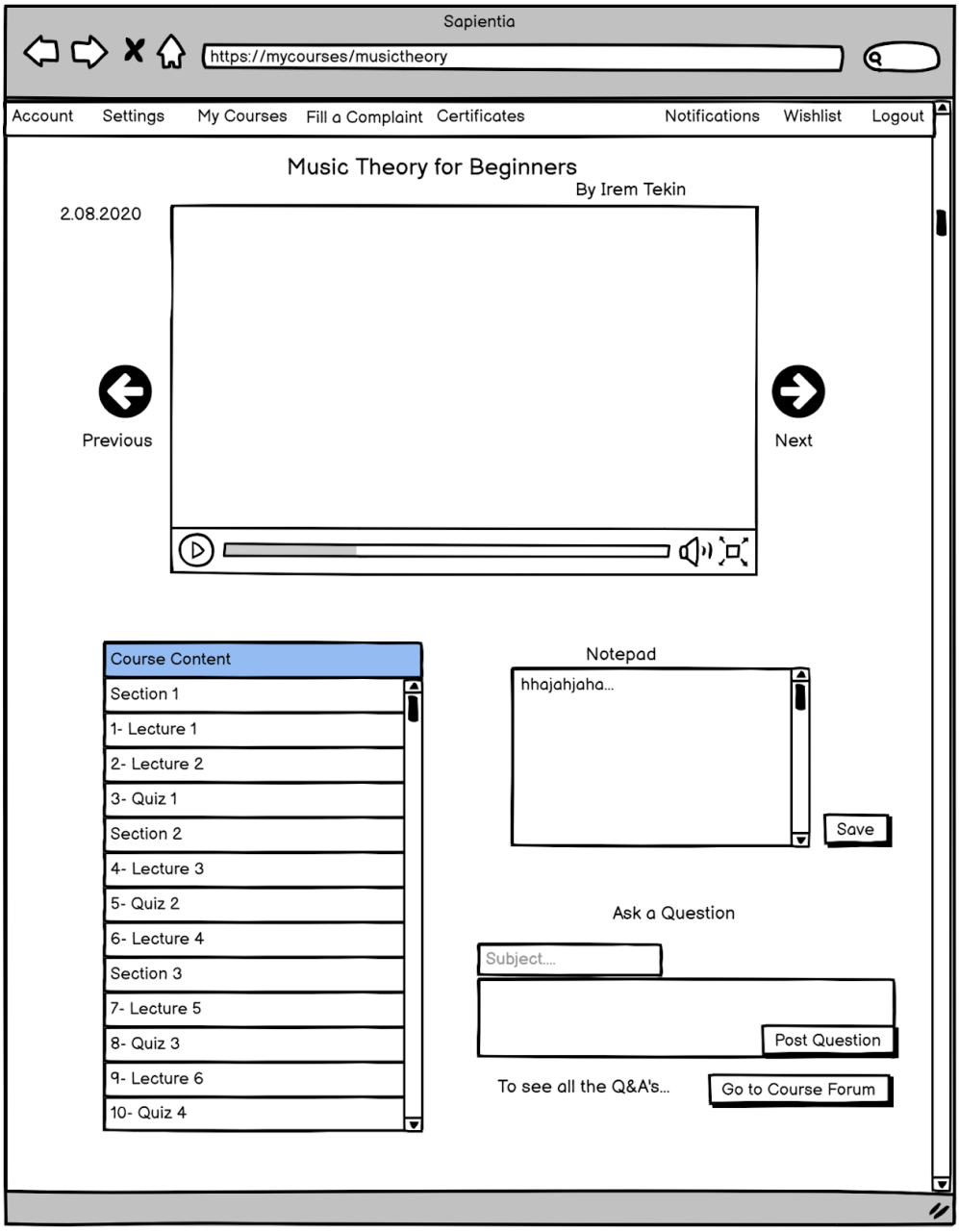


The users will be able to see the courses that they purchased from the **My Courses** button in the navigation bar. On that page, they will be able to see the Q&A section from the Forum button and go to the course's own page to continue watching the course.

Case : In the above example the student purchased 3 courses. The first course has a Zoom Q&A meeting conducted by the instructor where the student can join the meeting by the Join button and will be directed to Zoom.

```
SELECT course_name, course_photo, invite_link, start_date
FROM Course NATURAL JOIN Enrolls FULL OUTER JOIN Zoom_Session
WHERE SID = @SID
```

8. Course - Lecture



As shown in the **left corner**, course content is categorized by sections and under each section, there are quizzes and lectures. The students will also be able to ask a question to the forum and add notes to their notepad while watching a lecture.

Case: In the above example, the student is at Section **2** and watching **Lecture 3**. The student can press the next and see the next course content which is **Quiz 2** in this example. However, after the video is finished , the course will automatically skip to the next content.

```
SELECT lecture_content, content_num, date
        FROM Lecture
        WHERE CID = @CID AND content_num = @content_num

SELECT content_num, title
FROM Lecture

SELECT content_num, title
FROM Quiz


SELECT text
FROM Note NATURAL JOIN Student NATURAL JOIN Lecture
WHERE CID = @CID AND content_num = @content_num AND SID = @SID
```

Input: @note_text, @text, @subject_name

iii. Create notes on lectures (Visible only to user)

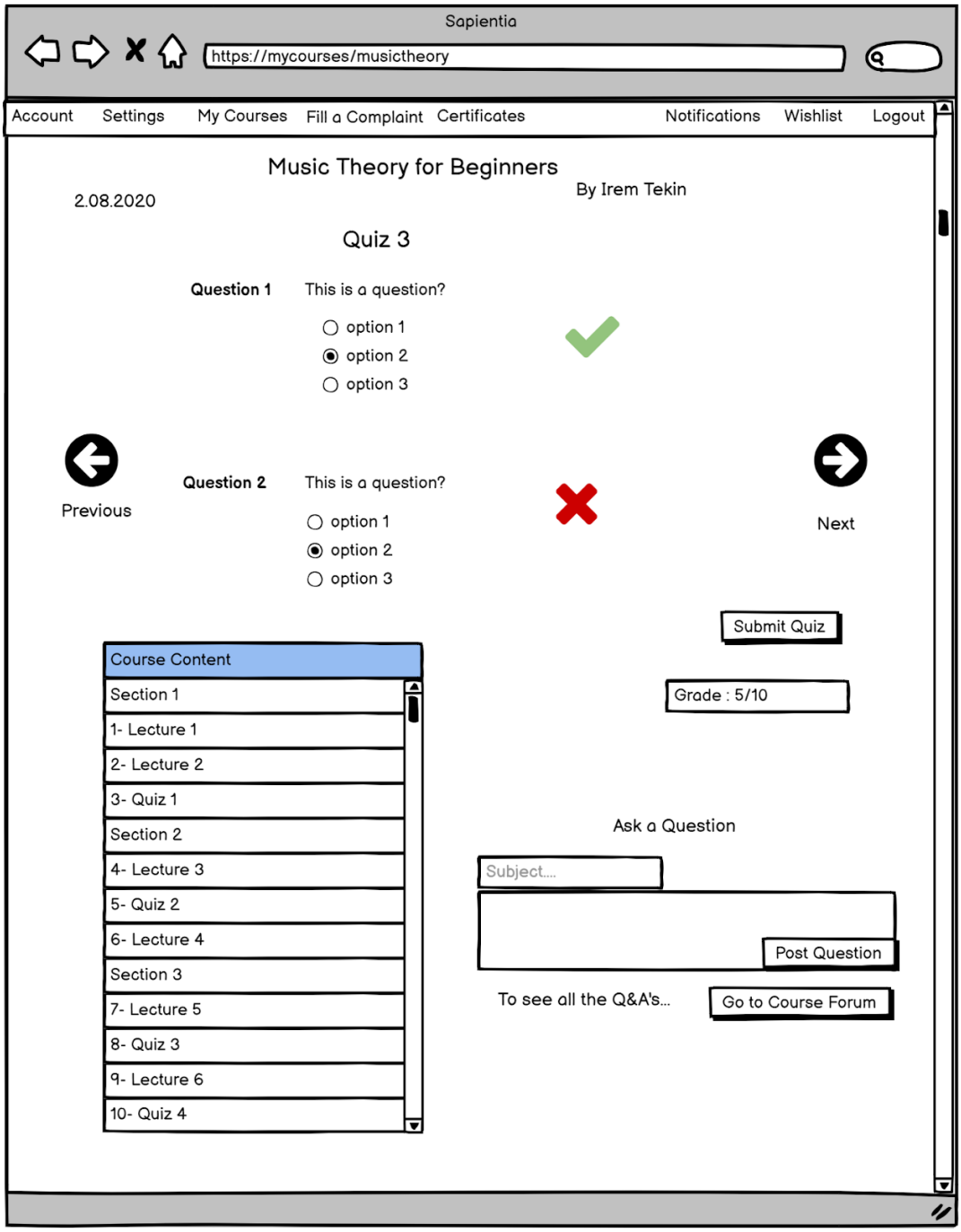
On Save button pressed:

```
INSERT INTO Note(SID, content_num, text)
VALUES(@SID,@content_num,@note_text)
```

On Post Question button pressed:

```
DECLARE @current_date date = getdate()
INSERT INTO QnA_Entry_Student(CID, SID, subject_name, text, date)
VALUES(@CID, @SID, @subject_name, @text, @current_date)
```

9. Course- Quiz



There will be quizzes in a course as well as lectures. As shown in the **left corner**, course content is categorized by sections and under each section, there are quizzes and lectures. The students will also be able to ask a question to the forum while they are solving the quiz.

Case: In the above example, the student is at **Section 3** and solving **Quiz 3**. The first question is answered correctly and the second is false. After solving the quiz, the student can press the next and see the next course content which is **Lecture 6** in this example.

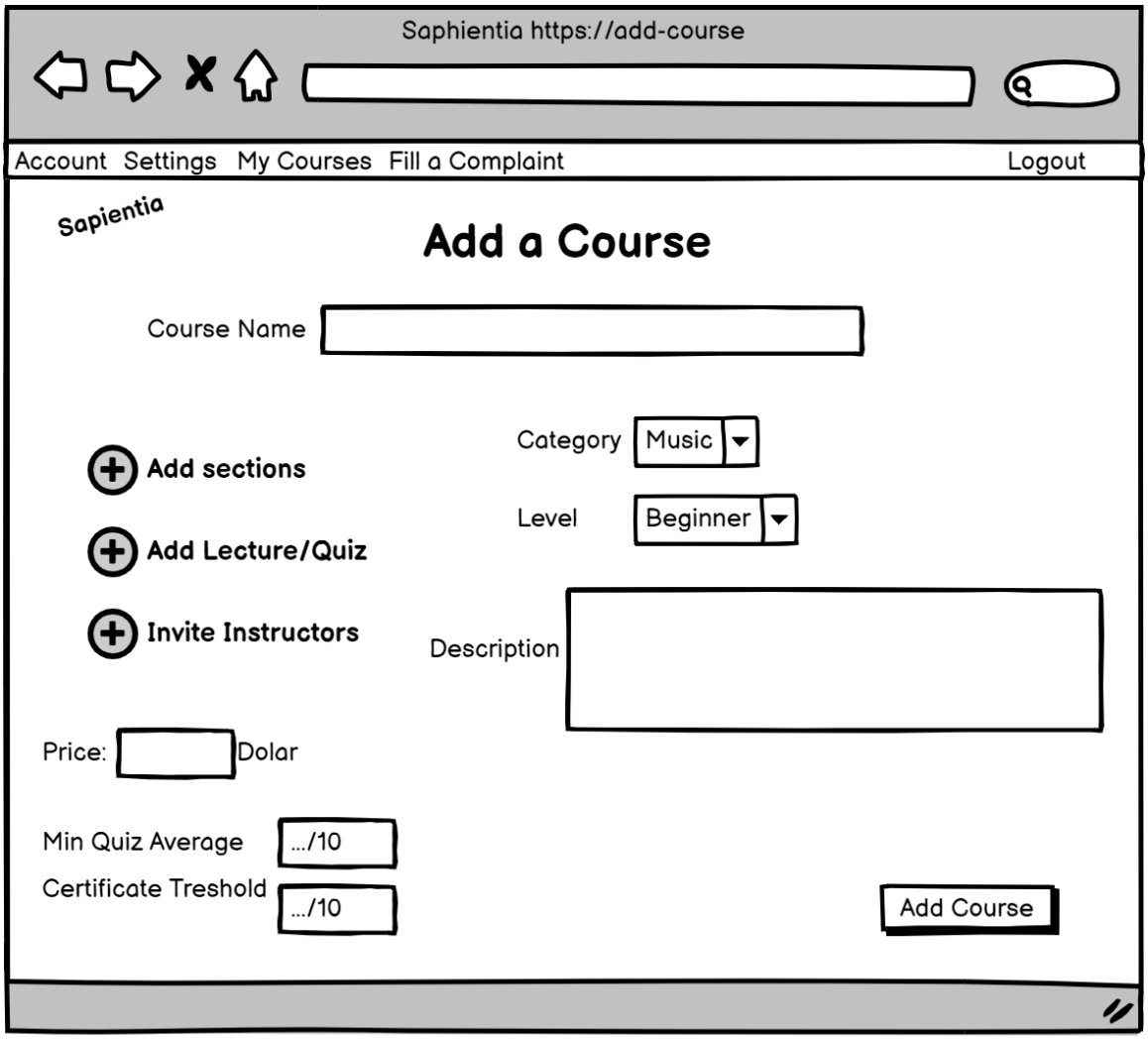
Inputs: @subject_name, @answer, @text

```
SELECT date, name, title, question_num, question_text, choice1, choice2, choice3
FROM Quiz_Question NATURAL JOIN Quiz NATURAL JOIN Course
WHERE CID = @CID AND content_num = @content_num
```

On Submit Quiz button pressed

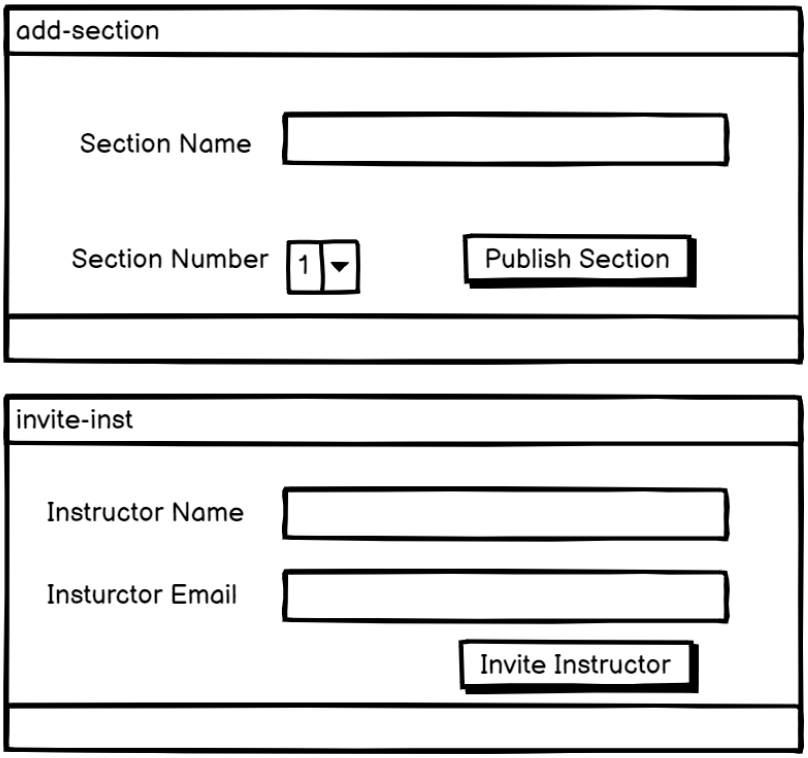
```
INSERT INTO Take_Quiz_Question(SID, CID, content_num, question_num, answer, isTrue)
VALUES (@SID, @CID, @content_num, @question_num, @answer, @isTrue)

INSERT INTO Take_Quiz(SID, CID, content_num, grade)
VALUES(@SID, @CID, @content_num, @grade)

SELECT grade
FROM Take_Quiz
WHERE SID = @SID AND CID = @CID AND content_num = @content_num


DECLARE @current_date date = getdate()
INSERT INTO QnA_Entry_Student(CID, SID, subject_name, text, date)
VALUES(@CID, @SID, @subject_name, @text, @current_date)
```

10. Add a Course



An instructor can add a course from the **Add Course Page .** From this page, the instructor can determine the sections of the course,the price, minimum quiz average for a student to receive a certificate, lectures or quizzes to the course, add description, choose the category or difficulty level of the course. This page is shown in order to explain additional quiz and invite instructor feature. SQL statements for this page is not given because of this, SQL statements for quiz feature are given in the following parts.

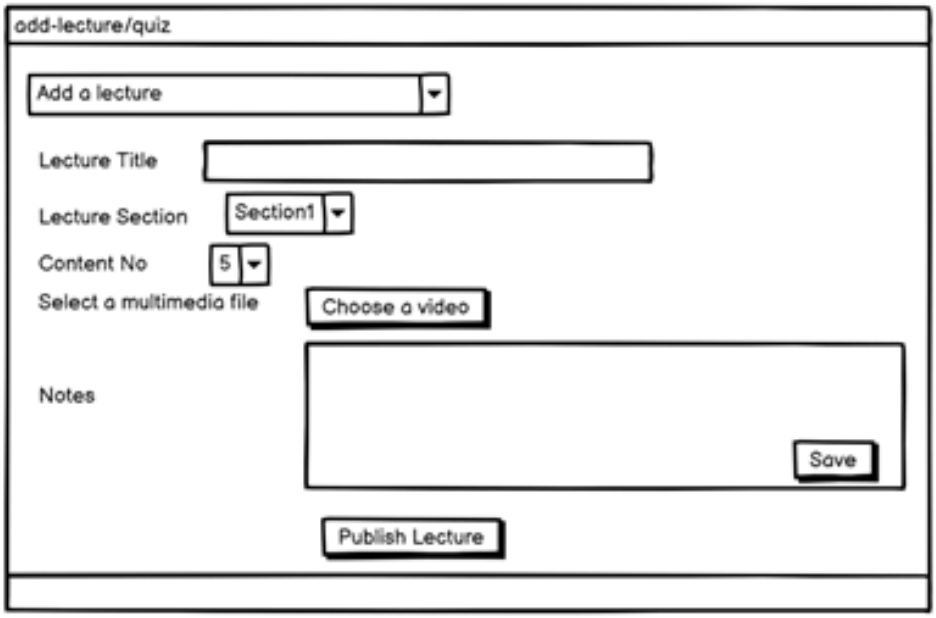
Add Section / Invite Another Instructor



While adding a course, as an additional feature, the instructor will be able to add sections to the course and add one more instructor to the course. This page doesn't have an SQL statement now because it is actually related to the publish course part (second part in functionality document). This page is shown in order to represent additional functionality invite instructor.

Case: In order to add an instructor to a course the instructor needs to send an invitation to the other instructor by using his/her name and email. To add a section to the course, the name of the section and the number of the section is needed.

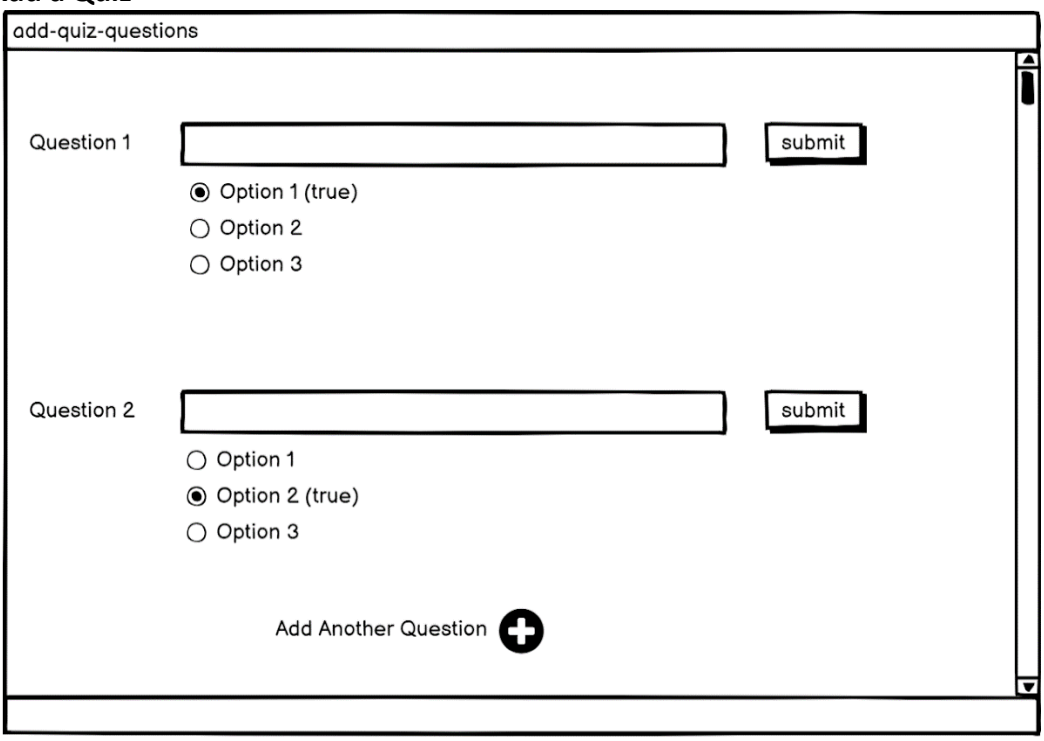
Add Lecture

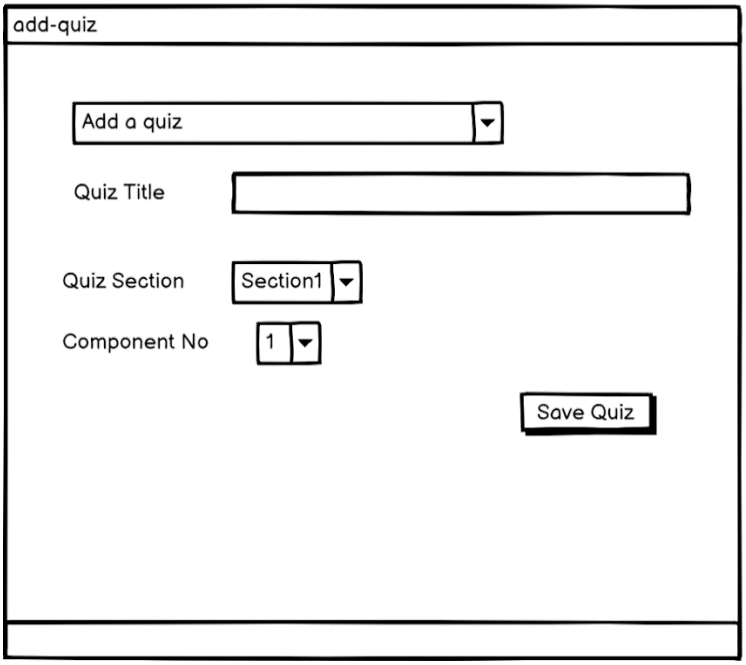


The instructor can either add a lecture or a quiz to the course while adding the course or editing the course. This page doesn't have an SQL statement now because it is actually related to the publish course part (second part in functionality document).

Case: In the example above, the instructor adds a lecture to the course. In order to add a lecture, the instructor needs to give a title, section and a component number to lecture. Component number indicates the order of the lecture among other lectures and quizzes. Then, the instructor needs to upload a video file and add additional notes to the lecture.

Add a Quiz





While adding or editing a course, instructors will be able to add quizzes to their course. First, they will choose whether they want to add a quiz or a lecture to their course. Then, they will give a title to their quiz, choose the section of the quiz and indicate the component number of the quiz. The component number shows the order of the quiz in the lecture/quiz list of the course.

Then they will be directed to the Edit Quiz Page where they can write as many questions as they want.

Case: In the above example adding a quiz to a course is shown.

Input: @title, @section, @content_num

On Save Quiz button pressed:

```
DECLARE @current_date date = getdate()
INSERT INTO Quiz (title, section, CID, content_num, date)
VALUES (@title, @section, @CID, @content_num, @current_date)
```

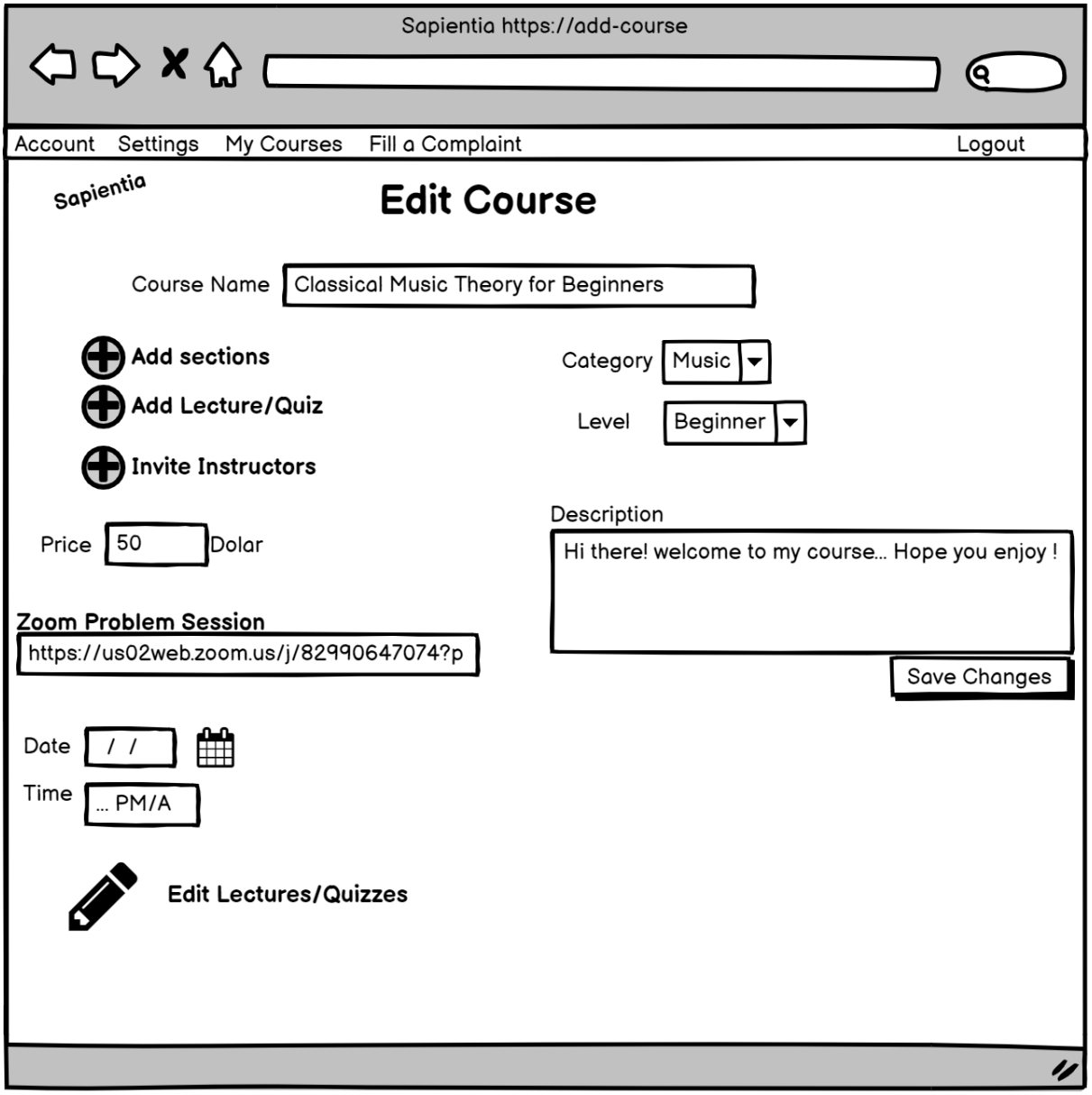
Add question

Input: @question_text, @choice1, @choice2, @choice3, @answer

On Save button pressed:

```
INSERT INTO Quiz_Question (CID, content_num, question_num, question_text, choice1,
choice2, choice3, answer)
VALUES ( @CID,@content_num @question_num, @question_text, @choice1, @choice2,
@choice3, @answer)
```

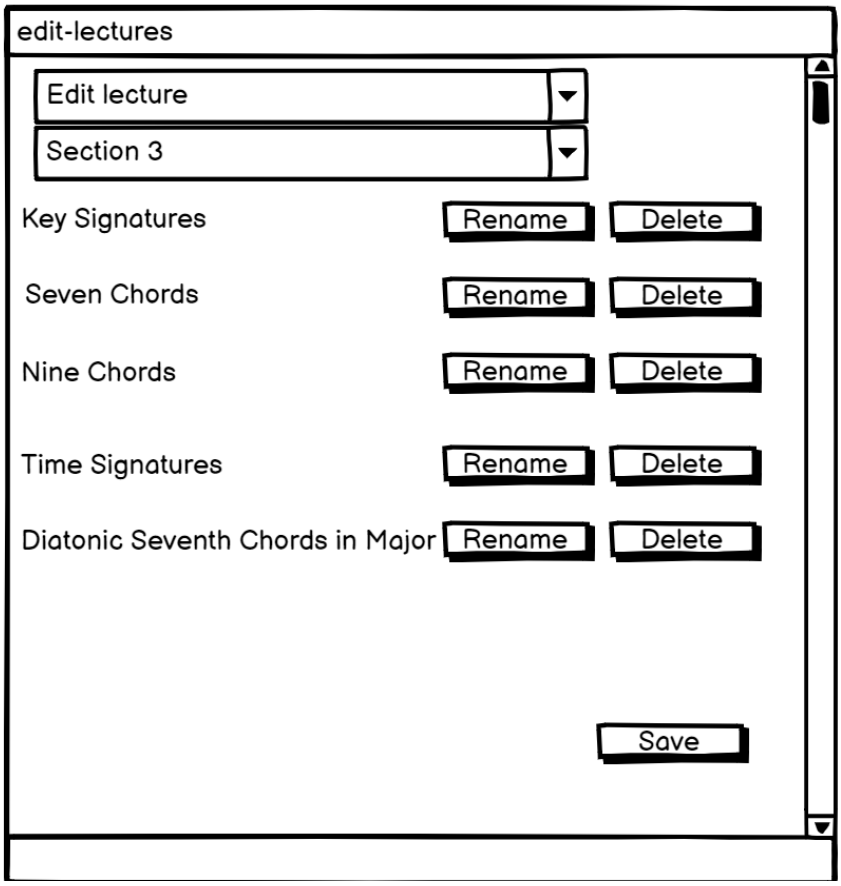
Edit a Course



When instructors press the **Edit Lectures/Quizzes** button in the edit course page, they will be directed to the **edit page** where they can choose to edit either quizzes or lectures from the combo box in the top right corner. This page doesn't have an SQL statement now because it is actually related to the publish course part (second part in functionality document).

Case: In the example above, the instructor wants to edit the quizzes in a course where the section of quiz is Section 2. The instructor can edit a quiz by deleting/adding questions to it, changing the questions or quiz' section . The instructor can also delete the quizzes from a section.

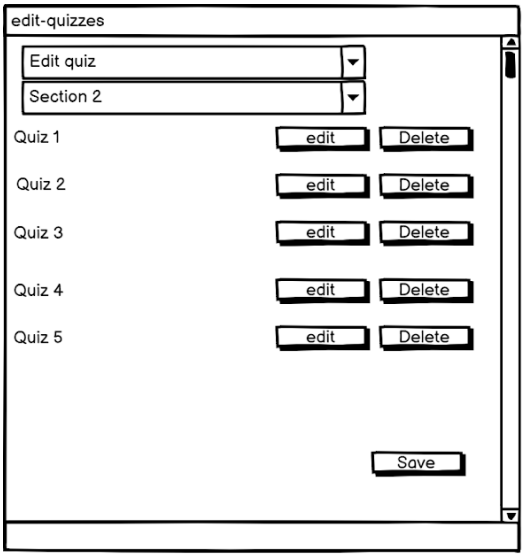
Edit Lectures

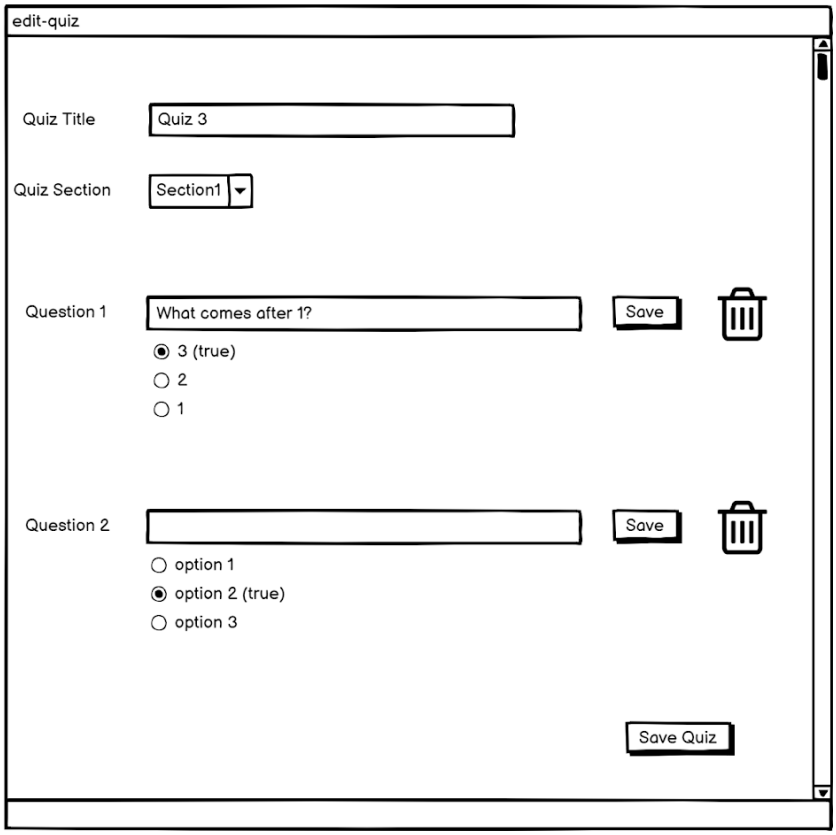


When instructors press the **Edit Lectures/Quizzes** button in the Edit Course Page, they will be directed to the **edit page** where they can choose to edit either quizzes or lectures from the combo box in the top right corner.

Case: In the example above, the instructor wants to edit lectures by either deleting them or renaming them. In the case of changing the lecture video, the instructor needs to first delete the video, then add a video from the Edit Course Page.

Edit Quizzes





When instructors press the **Edit Lectures/Quizzes** button in the edit course page, they will be directed to the **edit page** where they can choose to edit either quizzes or lectures from the combo box in the top right corner.

Case: In the example above, the instructor wants to edit the quizzes in a course where the section of quiz is Section 2. The instructor can edit a quiz by deleting/adding questions, changing the questions or quiz's section. The instructor can also delete the quizzes from a section.

Input: @section, @title, @question_text, @choice1, @choice2, @choice3, @answer

```
SELECT title
FROM Quiz
WHERE CID = @CID AND section = @section
```

Delete button pressed:

```
DELETE FROM Quiz
WHERE CID = @CID AND content_num = @content_num
```

Edit specific quiz page

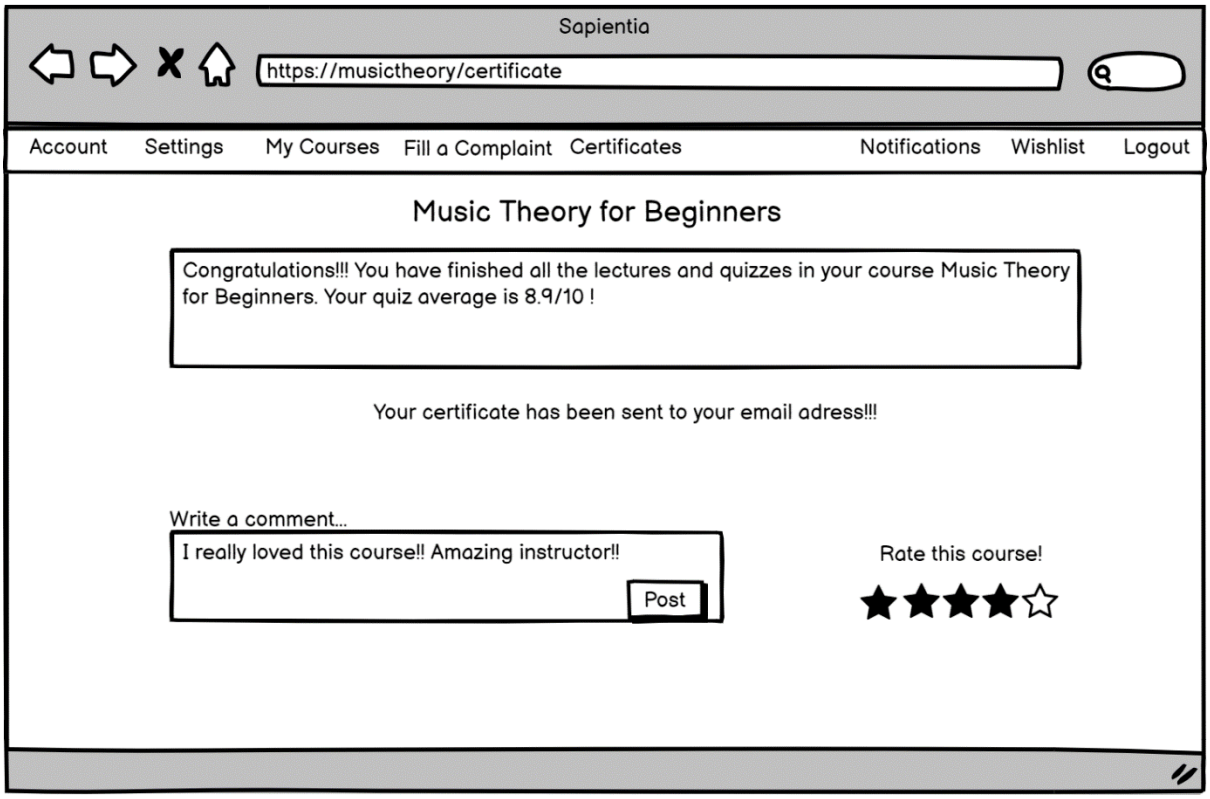
On Save Quiz button pressed:

```
UPDATE Quiz
SET title = @title, section = @section
WHERE CID = @CID AND content_num = @content_num
```

On Save button pressed:

```
UPDATE Quiz_Question
SET question_text = @question_text, choice1 = @choice1, choice2 = @choice2, choice3 =
@choice3, answer = @answer
```

11. Certificate



In order to finish a course student must watch all the lectures and have a quiz average above the threshold specified by the instructors. Then, they will receive a certificate from the website indicating that they finished the course. In addition, they will be able to rate the course and leave a comment about the course.

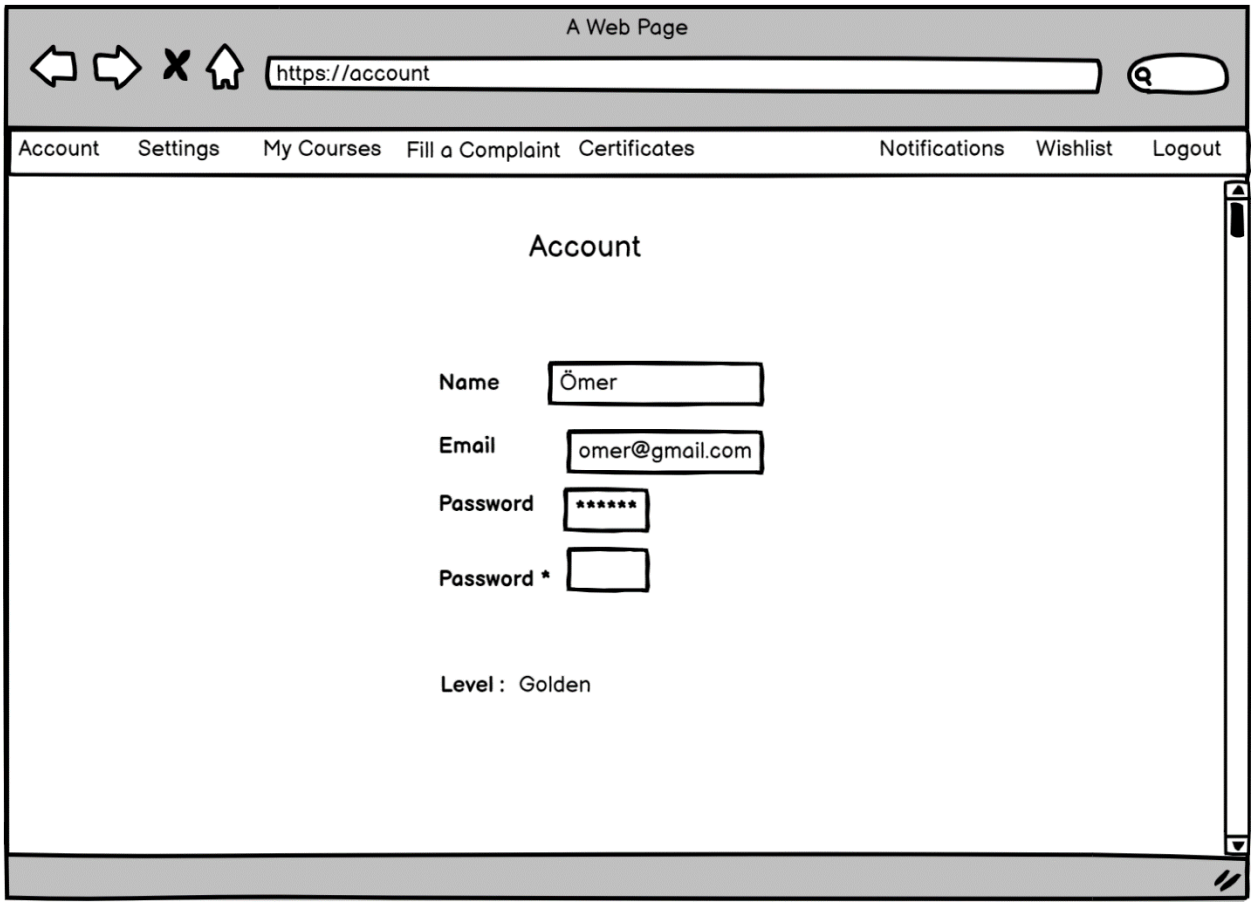
Case: In the above example, the student finished the course with the quiz average of 8.9/10, about to leave a comment to the course and rated the course as 4/5.

Inputs: @student_rating

```
SELECT COUNT(*)
FROM Lecture
WHERE CID = @CID

SELECT COUNT(*)
FROM Quiz
WHERE CID = @CID

SELECT COUNT(*)
FROM Take_Lecture NATURAL JOIN Student
```

```
WHERE CID = @CID AND isCompleted = 'true'

SELECT AVG(grade)
FROM Take_Quiz NATURAL JOIN Student
WHERE CID = @CID

SELECT quiz_threshold
FROM Course
WHERE CID = @CID

UPDATE Enrolls
SET rating = @student_rating
WHERE SID = @SID AND CID = @CID

SELECT rating
FROM Course
WHERE CID = @CID

WITH Temp(avg_rating) as
(SELECT CID, AVG(rating)
FROM Enrolls
GROUP BY CID
HAVING CID = @CID),
DECLARE @avg  NUMERIC(2,1)
SELECT @avg = avg_rating
FROM Temp

UPDATE Course
SET rating = @avg
```

12. Student Account



A student is a bronze member by default. Then will be upgraded to silver and golden member if he /she purchases 10 or 20 courses respectively.

A student can reach his/her account by pressing to the Account button in the navigation bar. In there, information about the student is listed.

Case: In the example above, the student is a golden member which means he has purchased 25 courses.

```
SELECT ( name, e_mail, password, membership_type )
FROM Student
WHERE SID = @SID
```